Name: Mark R. Fahey
Address: faheymr@ornl.gov

A Parallel Eigenvalue Routine for Complex Hessenberg Matrices

Mark R. Fahey
Joint Institute for Computational Sciences
Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6203

A code for computing the eigenvalues of a complex Hessenberg matrix is presented. This code computes the Schur decomposition of a complex Hessenberg matrix. Together with existing ScaLAPACK routines, the eigenvalues of dense complex matrices can be directly computed using a parallel QR algorithm.

This parallel complex Schur decomposition routine was developed to fill a void in the ScaLAPACK library and was based on the parallel real Schur decomposition routine already in ScaLAPACK. The real-arithmetic version was appropriately modified to make it work with complex arithmetic and implement a complex multiple bulge QR algorithm. This also required the development of new auxiliary routines that perform essential operations for the complex Schur decomposition, and that will provide additional linear algebra computation capability to the parallel numerical library community.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming—*distributed programming, parallel programming*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*eigenvalues*

General Terms: Parallel programming

Additional Key Words and Phrases: Parallel QR algorithm, Schur decomposition, ScaLAPACK, complex matrices, eigenvalue

1. INTRODUCTION

A parallel code that computes the Schur decomposition of a complex Hessenberg matrix is presented. Since it reduces the Hessenberg matrix to triangular form, the eigenvalues of the Hessenberg matrix are then readily available.

This code was created by converting the ScaLAPACK [Blackford et al. 1997] code `PDLAHQR` to a complex implementation. `PDLAHQR` computes the Schur decomposition of a real matrix in parallel using a multiple bulge QR algorithm. Therefore, this implementation, called `PZLAHQR`, also computes the Schur decomposition of a complex matrix in parallel using a multiple bulge strategy. Since the output from `PZLAHQR` is standard Schur form, `PZLAHQR` can be used in conjunction with existing ScaLAPACK routines to compute eigenvalues and eigenvectors of a general complex matrix in parallel. The name of this code and its auxiliary routines follow the ScaLAPACK convention.

The work by Henry, Watkins, and Dongarra [Henry et al. 1997] is the primary reference on the parallel nonsymmetric QR algorithm design. Their paper includes many details that will not be repeated here and should be referenced in addition to this work.

In Section 2, both the sequential single bulge and and multiple bulge QR algorithms are reviewed. In Section 3, highlights of the parallel nonsymmetric QR algorithm are presented from [Henry et al. 1997]. The conversion of the real parallel nonsymmetric QR algorithm to complex arithmetic is discussed in Section 4. Numerical accuracy and scalability results for `PZLAHQR` are presented in Section 5. Concluding remarks are given in Section 6.

2. SEQUENTIAL QR ALGORITHM REVIEW

The implicit shifted QR algorithm has been a successful serial method for computing the Schur decomposition

$$H = QTQ^H$$

where H is a Hessenberg matrix, Q is a unitary matrix, and T is an upper triangular matrix. The initial matrix H is assumed to be Hessenberg for simplicity since the reduction to Hessenberg form is a well understood problem and has been shown to parallelize well [Berry et al. 1994; Dongarra and Van de Geijn 1992].

The Schur decomposition is computed by iteratively applying orthogonal similarity transformations to the Hessenberg matrix H until it becomes upper triangular. This process, known as the QR algorithm, performs QR iterations implicitly by chasing *bulges* down the subdiagonals of the upper Hessenberg matrix H [Golub and Van Loan 1996; Watkins 1991]. Each iteration begins by choosing shifts that accelerate convergence and using them to form a bulge. The bulge is then chased down the subdiagonals to complete the iteration.

2.1 Single Bulge

The Francis double implicit shifted QR algorithm has long been the standard serial version for real matrices. One step of the Francis QR algorithm is presented in Figure 1.

Francis QR Step
$e \leftarrow \text{eig}(H(n-1:n, n-1:n))$
$x \leftarrow (H - e(1)I_n)(H - e(2)I_n)I_n(1:n, 1)$
Let $P_0 \in \mathbb{C}^{n\times n}$ be a Householder matrix
    such that $P_0 x$ is a multiple of $I_n(1:n, 1)$
$H \leftarrow P_0 H P_0$
for $i = 1, \ldots, n-2$
    If $i < n-2$
        Compute P_i so that $P_i H$ has zero
            $(i+2, i)$ and $(i+3, i)$ entries
    If $i = n-2$
        Compute P_i so that $(P_i H)_{n,i} = 0$
    Update $H \leftarrow P_i H_i P_i$
    Update $Q \leftarrow QP_i$
endfor

Fig. 1. Sequential Single Bulge Francis QR Step [Henry et al. 1997]

The Householder matrices in Figure 1 are unitary transforms of the form:

$$P_i = I - \tau v v^H \tag{1}$$

where $\tau \in \mathbb{C}$ and $1 \leq |\tau| \leq 2$ and $v \in \mathbb{C}^n$ [Lehoucq 1994].

The Francis QR step begins each iteration by choosing two shifts and using them to form a bulge of degree 2. The bulge is then chased from top to bottom to complete the iteration. The shifts are usually taken to be the eigenvalues of the 2×2 submatrix at the lower right; this is commonly referred to as the Wilkinson shift strategy. One could also use some larger number, say M, of shifts by computing the eigenvalues of the lower right hand submatrix or order M. This leads to the multiple bulge QR algorithm discussed in the next section.

2.2 Multiple Bulge QR Algorithm

To chase multiple bulges simultaneously, the double-shift strategy is inadequate because the shifts for the next iteration cannot be calculated until the current bulge has been chased to the bottom of the matrix. A multishift QR algorithm was proposed in [Bai and Demmel 1989] which had multiple single QR steps carried out simultaneously chasing one large bulge where the eigenvalues of the lower $M \times M$ submatrix are the shifts. This algorithm was found to have a flaw: it must compute many iterations before the first batch of eigenvalues is deflated, and the problem becomes exacerbated with increasing M. Watkins [Watkins 1994] then proposed chasing many small bulges instead. For example use M shifts to perform a batch of $M/2$ double steps, one after the other. In [Watkins and Elsner 1991], it was proved that this results in quadratic convergence like the double-shift QR algorithm. Each cycle of computing M shifts and chasing $M/2$ bulges is referred to as a *super-iteration*. One super-iteration of the general sequential multiple bulge algorithm is presented in Figure 2.

In Figure 2, the i index is similar in function to the i index in the algorithm shown in Figure 1. Since there are $M/2$ bulges, some start-up and wrap-up costs exist (see [Henry et al. 1997] for more details).

3. PARALLEL QR ALGORITHM

A parallel nonsymmetric QR algorithm for real matrices was implemented in the code `PDLAHQR` as part of ScaLAPACK. The algorithm used in `PDLAHQR` is similar to the LAPACK routine `DLAHQR`. However, unlike `DLAHQR`, instead of sending one double-shift through the largest unreduced submatrix, this algorithm sends multiple double-shifts. This allows all bulges to carry out up-to-date shifts and spaces them apart so that there can be parallelism across several processor row/columns. Another critical difference is that this algorithm applies multiple double-shifts in a block fashion, as opposed to `DLAHQR`, which applies one double-shift at a time. Note that the LAPACK code `ZHSEQR` is a multiple shift, single bulge QR algorithm implementation.

This is the approach taken in [Henry et al. 1997] where M shifts are obtained from the lower $M \times M$ submatrix, where M is a fairly large even number (say 40), and used to form $S = M/2$ bulges of degree two and chase them one after the other down the subdiagonal in parallel.

```
Multiple Bulge QR Super-Iteration
e ← eig(H(n − m + 1 : n, n − m + 1 : n))
for k = 0, ..., n − 6 + 2m
    for j = m, m − 2, m − 4, ..., 2
        i = k − 2j + 4
        if i < 0 then P_i = I
        if i = 0
            x ← (H − e(j−1)I_n)(H − e(j)I_n)I_n(1 : n, 1)
            Let P_i ∈ C^{n×n} be a Householder matrix
                such that P_i x is a multiple of I_n(1 : n, 1)
        if 1 ≤ i < n − 2
            Compute P_i so that P_i H has zero (i + 2, i)
                and (i + 3, i) entries
        if i = n − 2
            Compute P_i so that (P_i H)_{n,i} = 0
        if i > n − 2 then P_i = I
        H ← P_i H P_i
        Q ← Q P_i
    endfor
endfor
```

Fig. 2. Sequential Multiple Bulge QR Super-Iteration [Henry et al. 1997]

Details of this approach are discussed by Henry, Watkins, and Dongarra [Henry et al. 1997]; their key observations pertaining to parallelization are as follows:

—The most critical difference between serial and parallel implementations of the QR algorithm is that the number of bulges must be chosen to keep the processors busy. The bulges must be separated by at least a block, and remain synchronized, to ensure that each row/column of processors remains busy. Usually the block size must be large; otherwise there will be too much boundary communication.

—The overall logic can be kept similar to the well-tested QR algorithm. The super-iteration can be implemented to complete before new shifts are determined and another super-iteration is begun. Information about the "current" unreduced submatrix must remain global to all nodes.

—The Householder transforms are of size 3, which means they are specified by sending 3 data items. The latency associated with sending many such small messages would be ruinous, so the information from several (e.g., 30) Householder transformations is bundled in each message.

—If many bulges are being chased simultaneously, there may be several bulges per row or column of processors. In that case, latency can be reduced further by combining the information from all bulges in a given row or column into a single message.

This concludes a brief review of a parallel multiple bulge QR algorithm and the solver `PDLAHQR` based on this approach. In what follows, a complex version of the multiple bulge nonsymmetric QR algorithm is developed.

4. CONVERSION OF PDLAHQR TO PZLAHQR

This sections describes the changes involved in converting PDLAHQR to a complex implementation, namely PZLAHQR. Counterparts to the auxiliary codes associated with PDLAHQR were also converted.

In the following subsections, new auxiliary routines needed in the development of PZLAHQR are discussed as well as PZLAHQR itself.

Note that the double-shift strategy produces eigenvalues that appear one at a time or in pairs. For real matrices, it is important that any QR algorithm implementation couples a complex eigenvalue with its complex conjugate, which is also an eigenvalue, so that complex arithmetic can be avoided. In fact, most QR algorithm implementations take great care to produce complex conjugate eigenvalues whose real components are equal and whose imaginary components are equal in magnitude to full precision. However, for PZLAHQR to be presented later, each pair of eigenvalues it finds will not necessarily be a complex conjugate pair. Moreover, even if a complex matrix has complex conjugate eigenvalues, this code does not pair them up and consequently does not ensure that the real components are identical or that the imaginary components have the same magnitude.

The double-shift strategy reduces the Hessenberg matrix to quasi-triangular form with 2×2 blocks on the diagonal. These blocks are then further reduced to triangular form. Then, if necessary, the reduction transformation (rotator matrix) is applied to the appropriate matrix rows and columns.

4.1 2×2 Schur Decomposition

Previously, it was noted that the 2×2 blocks that are formed along the diagonal are further reduced to upper triangular form, i.e.,

$$\begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} c & -\bar{s} \\ s & \bar{c} \end{bmatrix} \begin{bmatrix} \hat{w} & \hat{x} \\ 0 & \hat{z} \end{bmatrix} \begin{bmatrix} \bar{c} & \bar{s} \\ -s & c \end{bmatrix}$$

A routine to compute this decomposition was developed and denoted as ZLANV2. This routine is used by PZLAHQR, discussed below, to apply the rotator matrix (composed of cosine c and sine s values) to the corresponding rows and columns of the matrix where the reduced 2×2 block is located. To be useful in a parallel implementation, another auxiliary routine was created to apply a rotator to a distributed matrix, described next.

4.2 Parallel Application of Rotator Matrix

The efficient application of a 2×2 rotator in parallel to two rows (or columns) of a distributed matrix requires careful attention. The ScaLAPACK approach involves a 2-D block-cyclic distribution of the matrix over a 2-D process grid. The rows to be acted on by the rotator may reside on different process rows as well as being distributed column-wise. Although each processor will have the rotator matrix, each processor will not have all the necessary matrix information. To efficiently address this issue, a work array for every process is required. Each process, if needed, will receive a copy of the remote information it needs to update its locally owned row(s). Then each process can apply the rotator to its row(s) and work array in serial using ZROT. At this point, the matrix rows have been updated and no further communication is needed. This process has been implemented in the

PBLAS-like routine PZROT.

4.3 Parallel Complex QR Algorithm

With the aforementioned support routines and several auxiliary routines in place, all that remains is PZLAHQR itself. To implement this, the real code PDLAHQR was converted to a complex implementation. The two most significant modifications (beyond those of converting real arithmetic to complex and modify how the code tests for "small" elements) were as follows:

(1) At the beginning of the QR step, if two consecutive small subdiagonals are found, update a subdiagonal element by a factor of $1 - \tau$ (see Equation 1.) This is in comparison to PDLAHQR where the update factor is always -1. This meant introducing new coupled send and receive BLACS calls so that an up-to-date copy of τ can be used on the appropriate process.

(2) When 2×2 blocks are formed along the diagonal, they are transformed to standard Schur form, which uses ZLANV2. Then the corresponding rows and columns must be updated by PZROT.

As in PDLAHQR, PZLAHQR calls its serial counterpart ZLAHQR to compute eigenvalues of submatrices.

5. NUMERICAL RESULTS

In this section, the accuracy and scalability of PZLAHQR is presented. The numerical tests were carried out on an SGI Origin 2000, IBM SP2, and IBM Power3 SMP at the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC); and an IBM Power4 p690 at Oak Ridge National Laboratory. See Table 1 for more information on each machine.

Table 1. Parallel computing platforms used to test PZLAHQR.

Machine	Location	Processor Speed MHz	Mflops/s per proc.	Processors Per node	Number Nodes	Peak Gflops/s
SGI Origin 2000	ERDC	195	390	2	64	49.9
IBM SP2	ERDC	135	540	1	255	137.7
IBM Power3 SMP	ERDC	222	888	8[a]	64	454.6
IBM Power4	ORNL	1300	5200	32	27	4492.8

[a]When the original timings experiments were conducted, a maximum of four MPI processes could be used per node.

All routines are implemented in Fortran 77, except for PZROT, which is coded in C. For the tests at ERDC MSRC, all the routines were compiled using optimization flags `-O3`, `-O3 -qarch=pwr2 -qtune=pwr2 -qmaxmem=-1`, and `-O3 -qarch=pwr3 -qtune=pwr3 -qmaxmem=-1` for the SGI Origin 2000, the IBM SP2, and the IBM Power3 SMP, respectively. For the tests at ORNL, all the routines were compiled using optimization flags `-O4 -qnoipa -qmaxmem=-1`. For runs on all IBM machines,

the environment variable MP_SHARED_MEMORY was set to YES. All tests were run during normal operation hours in a non-dedicated environment.

5.1 Accuracy of PZLAHQR

The testing (and timing) results were performed using a modified version of the ScaLAPACK testing software for PDLAHQR to call PZLAHQR instead. The testing routine tests a randomly generated complex Hessenberg matrix over multiple processor grids and over multiple block sizes.[1]

The test code checks

$$\frac{|A - UTU^H|}{|A|n\epsilon} < \text{TOL} \tag{2}$$

and

$$\frac{|I - UU^H|}{n\epsilon} < \text{TOL} \tag{3}$$

for all combinations of matrix sizes, processor grids, and block sizes. As an example of the tests conducted, the following shows one example input file to the test code.

Input deck:

```
'SCALAPACK NEP (Nonsymmetric Eigenvalue Problem) input file'
'MPI machine'
'NEP.out'               output file name (if any)
6                       device out
5                       number of problems sizes
1 6 10 50 250           Problem sizes
3                       number of block sizes
6 8 17                  values of block sizes
4                       number of process grids (ordered pairs of P & Q)
1 2 1 4                 values of P
1 2 3 1                 values of Q
20.0                    threshold
```

For the above input set and for all runs in the following sections where timings are presented, PZLAHQR passed the residual tests (2) and (3) where the tolerance was 20. Please consult the ScaLAPACK Installation Guide [Choi et al. 1995] for more information on input files for ScaLAPACK test routines.

5.2 Fixed Problem Size Scalability of PZLAHQR

All timing results are for the computation of the standard Schur form of a Hessenberg matrix with random entries. A two dimensional block-cyclic data decomposition was used with a blocking factor of 100. Only square processor grids were used, but this is not a requirement. Any type of rectangular processor grid may be used. Note that the amount of data is $\mathcal{O}(N^2)$, and the run-time on a single processor is $\mathcal{O}(N^3)$.

[1] See ScaLAPACK installation guide [Choi et al. 1995].

In Tables 2, 3, and 4 the execution time in seconds for computing the complete Schur decomposition on an SGI Origin 2000, an IBM Power3 SMP, and an IBM SP2, respectively, is reported. The first line in the tables reports the timings for `ZHSEQR` from the LAPACK library. `ZHSEQR` is the sequential multiple single-shift QR algorithm for complex Hessenberg matrices. The remaining lines show the timings for `PZLAHQR` for various square processor grids.

Table 2. Execution time in seconds to compute a complex Schur decomposition on an SGI Origin 2000.

Execution time in seconds - SGI Origin 2000						
Proc. grid	Matrix Order					
mp × np	500	1000	1500	2000	2500	3000
1	64	521	1740			
1 × 1	66	552	1849			
2 × 2	23	155	568	1229	2417	4097
3 × 3	16	91	287	707	1278	2312
4 × 4	14	60	180	383	793	1123

Table 3. Execution time in seconds to compute a complex Schur decomposition on an IBM Power3 SMP.

Execution time in seconds - IBM Power3 SMP						
Proc. grid	Matrix Order					
mp × np	500	1000	1500	2000	2500	3000
1	40	321	1108	2568		
1 × 1	39	318	1106	2577		
2 × 2	14	106	340	764	1574	2514
3 × 3	9	58	198	428	813	1399
4 × 4	8	36	106	238	436	700

Table 4. Execution time in seconds to compute a complex Schur decomposition on an IBM SP2.

Execution time in seconds - IBM SP2						
Proc. grid	Matrix Order					
mp × np	500	1000	1500	2000	2500	3000
1	54	670	1500	3514		
1 × 1	54	448	1502	3471		
2 × 2	21	154	521	1387	2546	4279
3 × 3	13	83	269	632	1417	1957
4 × 4	12	52	154	339	862	1088

The results in Tables 2, 3, and 4 show that for fixed N, `PZLAHQR` scales well as the number of processors increases. This is based on the observation that `PZLAHQR` yields timings comparable to `ZHSEQR` for the one process tests, and as the number of processes increases, the timings for `PZLAHQR` reduce in nearly linear fashion.

In addition, there are some timings that are missing from the tables for sequential runs. The missing data are due to memory limitations. The code written to test PZLAHQR creates one large array for which all matrix and vector storage is used. For the sequential cases, this array must be so large that it would require 64-bit addressing to access the entire array. The Basic Linear Algebra Communication Subprograms (BLACS) are built with MPI as the underlying communication interface on the machines tested. At the time these tests were conduction, only the SGI Origin 2000 supported 64-bit compilation of MPI programs.

5.3 Scaled Problem Size Scalability of PZLAHQR

Next, the scalability of PZLAHQR is further investigated by calculating speedup and efficiency ratings based on the computed aggregate megaflop rate as the problem size and the number of processors increase. Assume that the flop count to compute the Schur decomposition is $18N^3$, where N is the order of the matrix [Blackford et al. 1997].

Let *efficiency* with respect to megaflop rate be defined as

$$E_F = \frac{M_p}{PM_s}$$

where M_p is the megaflop rate on P processors and M_s is the serial megaflop rate. The *speedup* with respect to megaflop rate

$$S_F = PE_F$$

is the factor by which execution time is reduced on P processors.

In Table 5, the time in seconds to compute the Schur decomposition for increasing larger problems and larger processor grids is shown. Notice that the efficiency

Table 5. Speedups and efficiencies based on the megaflop rate.

Proc. grid		IBM Power3				IBM Power4			
mp × np	N	Time	Mflops/s	S_F	E_F	Time	Mflops/s	S_F	E_F
1 × 1	2000	2577	56	1.0	1.00	842	171	1.0	1.00
2 × 2	4000	5975	193	3.5	0.86	1860	619	3.6	0.90
3 × 3	6000	13550	287	5.1	0.57	2465	1577	9.2	1.02
4 × 4	8000	12526	736	13.2	0.82	4282	2152	12.6	0.79
5 × 5	10000	23712	759	13.6	0.54	5389	3340	19.5	0.78
6 × 6	12000	18078	1720	30.8	0.85	5350	5814	34.0	0.94
8 × 8	16000	24471	3013	53.9	0.84	9014	8179	47.8	0.75

ratings are much higher for the processor grids that are evenly divisible by four in Table 5. In fact, the megaflop rate per processor for the processor grids divisible by four stays approximately constant around 47.

At the time the tests were conductioned, the IBM Power3 SMP had a limit of 4 MPI processes per node. The balancing of MPI processes for process grids not divisible by 4 was an issue. For example, with a 3×3 processor grid, an efficiency of 0.73 is obtained if the machine is set to use three MPI processes on three nodes instead of a four-four-one setup on three nodes that had an efficiency of 0.57 (as

shown in Table 5). Thus, the variable efficiencies are a by-product of the hardware, not the code.

The data in Table 5 are also displayed in Figure 3 and clearly show linear scalability. The exception is processor grids not divisible by four on the IBM Power3.

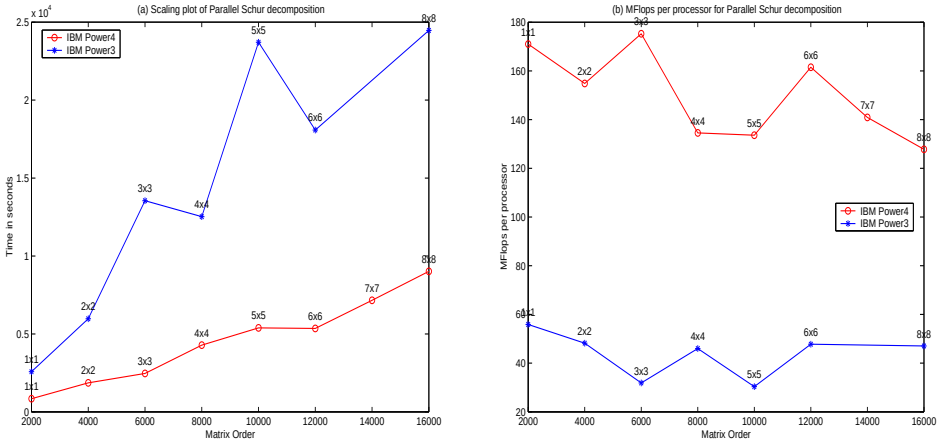

Fig. 3. Schur decomposition of a complex Hessenberg matrix using `PZLAHQR` using increasing larger square processor grids as the order of the matrix increases. (a) Time in seconds to compute Schur decomposition. (b) MFlops/s per processor.

6. CONCLUDING REMARKS

A parallel complex Schur decomposition routine `PZLAHQR` has been implemented based on the ScaLAPACK code `PDLAHQR`. Results were shown for `PZLAHQR` and showed that this routine scales nicely with the number of processors. Several auxiliary subroutines were developed to support this routine that will be useful outside the scope of `PZLAHQR`.

These codes have been added to the ScaLAPACK library starting with release 1.7. An updated version is in progress.

ACKNOWLEDGMENTS

The author thanks Greg Henry for his advice early in this project. The author also thanks Dan Duffy, William Ward, and the referees for their comments.

Parallel runs were done on the SGI Origin 2000, IBM SP2, and IBM Power3 SMP at the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC) and the IBM Power 4 at Oak Ridge National Lab (ORNL).

This research was supported by the U.S. Department of Energy, Office of Basic Energy Sciences at Oak Ridge National Laboratory; managed for the U.S. DOE by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725. This work was also

supported in part by a grant of computer time from the DoD High Performance Computing Modernization Program at the ERDC MSRC, Vicksburg, MS.

REFERENCES

Bai, Z. and Demmel, D. 1989. On a block implementation of Hessenberg multishift QR iteration. *Int. J. High Speed Comput. 1*, 97–112. Also Argonne National Laboratory Technical Report ANL-MCS-TM-127, 1989.

Berry, M. W., Dongarra, J. J., and Kim, Y. 1994. A highly parallel algorithm for the reduction of a nonsymmetric matrix to block upper-Hessenberg form. Computer Sciences Dept. Technical Report CS-94-221 (Feb.), University of Tennessee, Knoxville, TN. LAPACK Working Note #68.

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. 1997. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA.

Choi, J., Demmel, J., Dhillon, I., Dongarra, J., Ostrouchov, S., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. 1995. Installation guide for ScaLAPACK. Computer Sciences Dept. Technical Report CS-95-280 (March), University of Tennessee, Knoxville, TN. LAPACK Working Note #93.

Dongarra, J. J. and Van de Geijn, R. 1992. Reduction to condensed form on distributed memory architectures. *Parallel Computing 18*, 973–982.

Golub, G. H. and Van Loan, C. F. 1996. *Matrix Computations* (Third ed.). The Johns Hopkins University Press, Baltimore, MD.

Henry, G., Watkins, D., and Dongarra, J. 1997. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. Computer Sciences Dept. Technical Report CS-97-352 (March), University of Tennessee, Knoxville, TN. LAPACK Working Note #121.

Lehoucq, R. 1994. The computation of elementary unitary matrices. Computer Sciences Dept. Technical Report CS-94-233, University of Tennessee, Knoxville, TN. LAPACK Working Note #72.

Watkins, D. S. 1991. *Fundamentals of Matrix Computations*. John Wiley and Sons, New York, NY.

Watkins, D. S. 1994. Shifting strategies for the parallel QR algorithm. *SIAM J. Sci. Comput. 15*, 953–958.

Watkins, D. S. and Elsner, L. 1991. Convergence of algorithms of decomposition type for the eigenvalue problem. *Lin. Alg. Appl. 143*, 19–47.